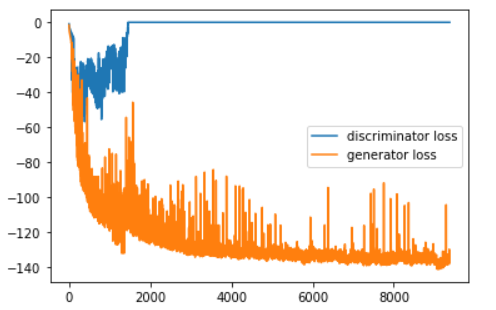
NN for Images – Ex3

David Guedalia

1. **Practical Task**

**GAN**

* 1. **Loss Saturation**
* On this section I followed the DCGAN architecture, I made a few changes of this architecture that I found improve the performances.
* I trained 3 steps on the discriminator per one single iteration of the generator.
* I implemented the generator with layers of conveTranspose, ReLU and batchNorm where I used Sigmoid ad the last layer.
* I implemented the discriminator with layers of conve, LeakyReLU and batchNorm where I used Sigmoid ad the last layer, I used LeakyReLU as I saw that it make the generator better results.

**תמונה שמכילה שוג'י, בניין, מרוצף

התיאור נוצר באופן אוטומטי1.1.1 The original GAN cross entropy**

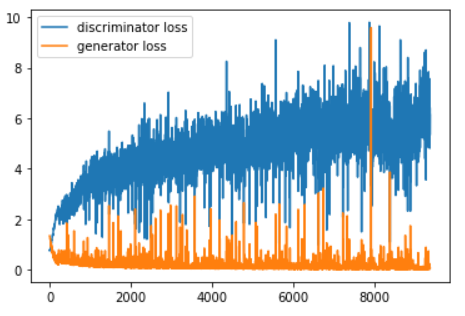
The results were bad, and as can see it didn’t learn, I can explain it as the discriminator converge much faster than the generator, and the discriminator have very small gradients and that’s why he can't learn well.

**1.1.2 The Non-Saturating BCE:**

We can see much better results, and even the best results from the three options, I think its because the loss maximize the likelihood of the discriminator that will be wrong –(for training the generator).

תמונה שמכילה חץ

התיאור נוצר באופן אוטומטי

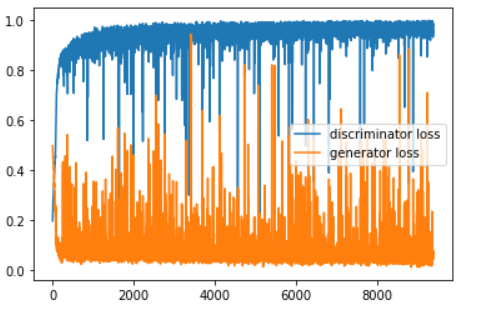


**1.1.2 The Least Squares Loss**

This loss also achives very good results, although it is not better then the non-saturating but we can see a very good results.

תמונה שמכילה חץ

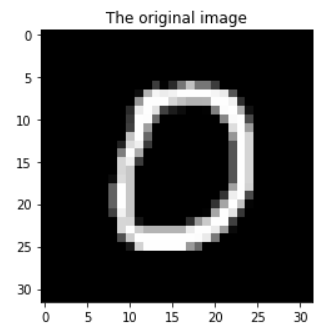
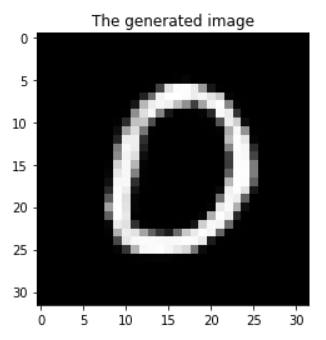
התיאור נוצר באופן אוטומטי

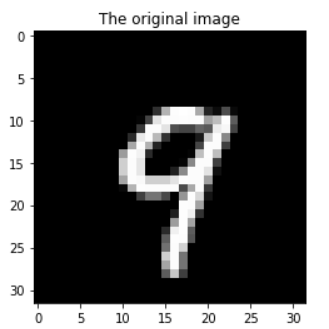
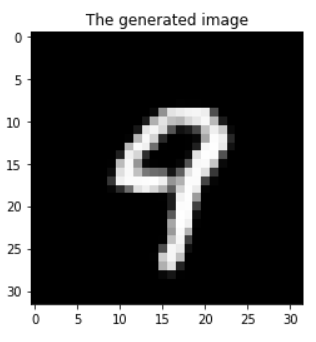


* 1. **Model Inversion**

In this section for given a picture I optimize over z and I used the MSE, means to find for image the vector z that the generator make the image.

I used images from the test, as we can see the inverse image looks like the original image, we can explain it as the generator were trained on similar images, also I found that the typical images gives better performance.





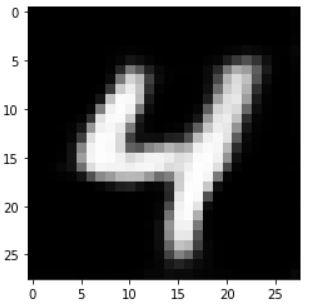
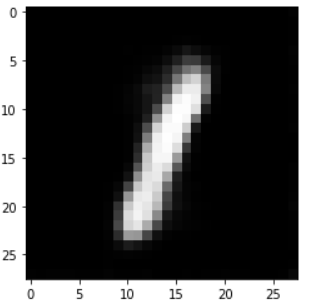
**Non-Adversarial Generative Networks**

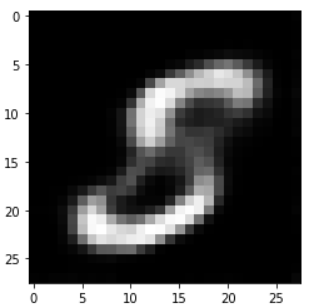
**2.1 Wasserstein Auto-Encoders**

In this section I reused code from the last exercise, I changed the loss, and add to the loss that latent space from the encoder to be close to the standard gaussian distribution.  
Basically, I used two functions of loss one for the decoder and the other for the encoder, and loss it between the gaussian of the encoder and the standard gaussian distribution.

From the results we can conclude that the decoder does not have good performance on the standard gaussian distribution comparing to the generator.

I can explain it because the generator is trained with normal distribution, and the autoencoder is not trained on random normal distribution



****

* 1. **Interpolation**

As we can see the Wasserstein autoencoder gives more elegant transitions, although the normal AE from the last exercise gives better digits and more clearly.

